# Distributed Big Data Analytics
# Efficient First Order Inductive Learner on Spark

Georgiy Shurkhovetskyy, Eskender Haziiev
Supervisors: Hajira Jabeen, Gezim Sejdiu

University of Bonn, 2017

## Problem Statement

**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.

## Problem Statement

**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.
**Definition 2**. A clause body is a conjunction of literals.

## Problem Statement

**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.
**Definition 2**. A clause body is a conjunction of literals.
**Definition 3**. A clause head is predicate.

## Problem Statement

**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.
**Definition 2**. A clause body is a conjunction of literals.
**Definition 3**. A clause head is predicate.
**Definition 4**. A Horn clause consists of a head and a body:

$$P \leftarrow L_1, L_2, ..., L_n. \tag{1}$$

**Definition 5**. A learning rule for predicate $P$ is a collection of Horn clauses each with head $P$.
The predicates can be defined *extensionally* as a list of tuples for which the predicate is true, or *intensionally* as a set of Horn clauses.

## Problem Statement

**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.
**Definition 2**. A clause body is a conjunction of literals.
**Definition 3**. A clause head is predicate.
**Definition 4**. A Horn clause consists of a head and a body:

$$P \leftarrow L_1, L_2, ..., L_n. \tag{1}$$

**Definition 5**. A learning rule for predicate $P$ is a collection of Horn clauses each with head $P$.
The predicates can be defined *extensionally* as a list of tuples for which the predicate is true, or *intensionally* as a set of Horn clauses.
**Definition 6**. $k$-tuple $< a_1, a_2, \ldots, a_k >$ is a finite sequence of $k$ constants.

## Problem Statement

**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.
**Definition 2**. A clause body is a conjunction of literals.
**Definition 3**. A clause head is predicate.
**Definition 4**. A Horn clause consists of a head and a body:

$$P \leftarrow L_1, L_2, ..., L_n. \tag{1}$$

**Definition 5**. A learning rule for predicate $P$ is a collection of Horn clauses each with head $P$.
The predicates can be defined *extensionally* as a list of tuples for which the predicate is true, or *intensionally* as a set of Horn clauses.
**Definition 6**. $k$-tuple $< a_1, a_2, \ldots, a_k >$ is a finite sequence of $k$ constants.
**Definition 7**. A tuple satisfies a learning rule if it satisfies one of the Horn clauses of this rule.
For every relation a set of the $\oplus$tuples which belong to this relation is given. For a target relation a set of $\oplus$tuples is also given. A set of the $\ominus$tuples not belonging to the target relation may be given. The statement is introduced: if some tuple is not included in set $\oplus$tuples then it is $\ominus$tuple.

## Problem Statement

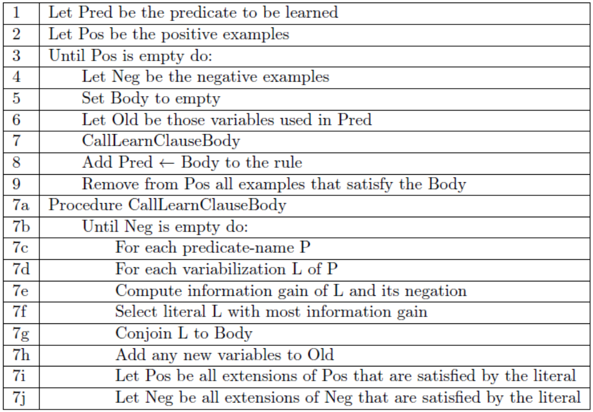
**Definition 1**. A literal $L$ may be predicate $P$ or $\neg P$.
**Definition 2**. A clause body is a conjunction of literals.
**Definition 3**. A clause head is predicate.
**Definition 4**. A Horn clause consists of a head and a body:

$$P \leftarrow L_1, L_2, ..., L_n. \tag{1}$$

**Definition 5**. A learning rule for predicate $P$ is a collection of Horn clauses each with head $P$.
The predicates can be defined *extensionally* as a list of tuples for which the predicate is true, or *intensionally* as a set of Horn clauses.
**Definition 6**. $k$-tuple $< a_1, a_2, \ldots, a_k >$ is a finite sequence of $k$ constants.
**Definition 7**. A tuple satisfies a learning rule if it satisfies one of the Horn clauses of this rule.
For every relation a set of the $\oplus$tuples which belong to this relation is given. For a target relation a set of $\oplus$tuples is also given. A set of the $\ominus$tuples not belonging to the target relation may be given. The statement is introduced: if some tuple is not included in set $\oplus$tuples then it is $\ominus$tuple.
**Problem statement:** Using Spark framework realize FOIL to find a learning rule as a set of Horn clauses for the target relation that consistent with given positive examples and not cover any given negative examples.

## Approach

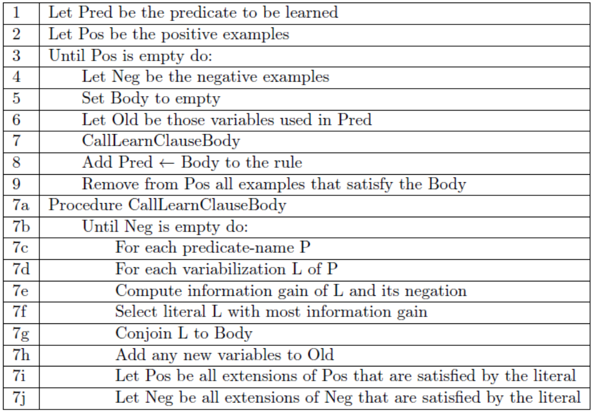
Foil algorithm is realized using Scala in Spark framework.



| | |
|---|---|
| 1 | Let Pred be the predicate to be learned |
| 2 | Let Pos be the positive examples |
| 3 | Until Pos is empty do: |
| 4 | Let Neg be the negative examples |
| 5 | Set Body to empty |
| 6 | Let Old be those variables used in Pred |
| 7 | CallLearnClauseBody |
| 8 | Add Pred $\leftarrow$ Body to the rule |
| 9 | Remove from Pos all examples that satisfy the Body |
| 7a | Procedure CallLearnClauseBody |
| 7b | Until Neg is empty do: |
| 7c | For each predicate-name P |
| 7d | For each variabilization L of P |
| 7e | Compute information gain of L and its negation |
| 7f | Select literal L with most information gain |
| 7g | Conjoin L to Body |
| 7h | Add any new variables to Old |
| 7i | Let Pos be all extensions of Pos that are satisfied by the literal |
| 7j | Let Neg be all extensions of Neg that are satisfied by the literal |

## Approach

Foil algorithm is realized using Scala in Spark framework.



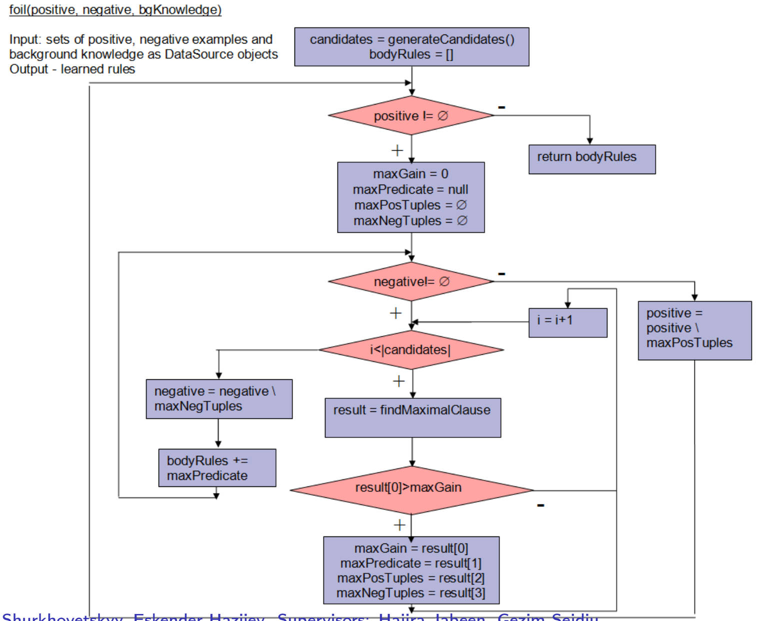
| | |
|---|---|
| 1 | Let Pred be the predicate to be learned |
| 2 | Let Pos be the positive examples |
| 3 | Until Pos is empty do: |
| 4 | Let Neg be the negative examples |
| 5 | Set Body to empty |
| 6 | Let Old be those variables used in Pred |
| 7 | CallLearnClauseBody |
| 8 | Add Pred $\leftarrow$ Body to the rule |
| 9 | Remove from Pos all examples that satisfy the Body |
| 7a | Procedure CallLearnClauseBody |
| 7b | Until Neg is empty do: |
| 7c | For each predicate-name P |
| 7d | For each variabilization L of P |
| 7e | Compute information gain of L and its negation |
| 7f | Select literal L with most information gain |
| 7g | Conjoin L to Body |
| 7h | Add any new variables to Old |
| 7i | Let Pos be all extensions of Pos that are satisfied by the literal |
| 7j | Let Neg be all extensions of Neg that are satisfied by the literal |

To select literal with greatest positive gain:

$$gain(L_i) = n_i^{\oplus\oplus} \cdot (I(T_i) - I(T_{i+1})), \quad I(T_i) = -\log_2(n_i^{\oplus}/(n_i^{\oplus} + n_i^{\ominus})), \qquad (2)$$

$n_i^{\oplus}$ — a number of $\oplus$tuples in $T_i$, $n_i^{\ominus}$ — a number of $\ominus$tuples in $T_i$,
$n_i^{\oplus\oplus}$ — a number of the $\oplus$tuples in $T_i$ represented by one or more tuples in $T_{i+1}$.

# Implementation


foil(positive, negative, bgKnowledge)
Input: sets of positive, negative examples and background knowledge as DataSource objects
Output - learned rules
candidates = generateCandidates()
bodyRules = []
positive != ∅
-
return bodyRules
+
maxGain = 0
maxPredicate = null
maxPosTuples = ∅
maxNegTuples = ∅
negative!= ∅
-
positive = positive \ maxPosTuples
+
i = i+1
i<|candidates|
negative = negative \ maxNegTuples
bodyRules += maxPredicate
+
result = findMaximalClause
result[0]>maxGain
-
+
maxGain = result[0]
maxPredicate = result[1]
maxPosTuples = result[2]
maxNegTuples = result[3]

## Evaluation

Evaluation metrics: $\rho_k(A_1, A_2) = |n_k(A_1) - n_k(A_2)|,\ k = 1, 2, 3.$

Indicators: $n_1(A) = \frac{n^{\oplus}_{\text{covered}}(A)}{n^{\oplus}_{\text{given}}}$, $n_2(A) = \frac{n^{\ominus}_{\text{covered}}(A)}{n^{\ominus}_{\text{given}}}$, $n_3(A) = \mathrm{runtime}(A)$.

## Evaluation

Evaluation metrics: $\rho_k(A_1, A_2) = |n_k(A_1) - n_k(A_2)|$, $k = 1, 2, 3$.

Indicators: $n_1(A) = \frac{n^{\oplus}_{\mathrm{covered}}(A)}{n^{\oplus}_{\mathrm{given}}}$, $n_2(A) = \frac{n^{\ominus}_{\mathrm{covered}}(A)}{n^{\ominus}_{\mathrm{given}}}$, $n_3(A) = \mathrm{runtime}(A)$.

Experiment 1. (a), results: $daughter(X_1, X_2) \leftarrow female(X_1) parent(X_2, X_1)$; covered $\oplus$ examples: (Emily, Tom), (Mary, Ann); covered $\ominus$ examples: $\emptyset$.

## Evaluation

Evaluation metrics: $\rho_k(A_1, A_2) = |n_k(A_1) - n_k(A_2)|,\ k = 1, 2, 3.$

Indicators: $n_1(A) = \frac{n^{\oplus}_{\text{covered}}(A)}{n^{\oplus}_{\text{given}}}$, $n_2(A) = \frac{n^{\ominus}_{\text{covered}}(A)}{n^{\ominus}_{\text{given}}}$, $n_3(A) = \mathrm{runtime}(A)$.

Experiment 1. (a), results: $daughter(X_1, X_2) \leftarrow female(X_1)parent(X_2, X_1)$; covered $\oplus$ examples: (Emily, Tom), (Mary, Ann); covered $\ominus$ examples: $\emptyset$.

Experiment 2. (b), results: $daughter(X_1, X_2) \leftarrow female(X_1)parent(X_2, X_1)$; covered $\oplus$ examples: (Opra, Ann), (Kate, Ann), (Lina, Kate), (Gail, Hovard); covered $\ominus$ examples: $\emptyset$.

## Evaluation

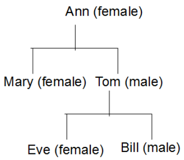
Evaluation metrics: $\rho_k(A_1, A_2) = |n_k(A_1) - n_k(A_2)|,\ k = 1, 2, 3.$

Indicators: $n_1(A) = \frac{n^{\oplus}_{\text{covered}}(A)}{n^{\oplus}_{\text{given}}},\ n_2(A) = \frac{n^{\ominus}_{\text{covered}}(A)}{n^{\ominus}_{\text{given}}},\ n_3(A) = \text{runtime}(A).$

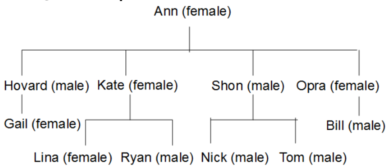
Experiment 1. (a), results: $daughter(X_1, X_2) \leftarrow female(X_1)parent(X_2, X_1)$; covered $\oplus$ examples: (Emily, Tom), (Mary, Ann); covered $\ominus$ examples: $\emptyset$.

Experiment 2. (b), results: $daughter(X_1, X_2) \leftarrow female(X_1)parent(X_2, X_1)$; covered $\oplus$ examples: (Opra, Ann), (Kate, Ann), (Lina, Kate), (Gail, Hovard); covered $\ominus$ examples: $\emptyset$.

Experiment 3. (b), results: $son(X_1, X_2) \leftarrow male(X_1)parent(X_2, X_1)$; covered $\oplus$ examples: (Ryan, Uve), (Bill, Opra), (Hovard, Ann), (Nick, Kate), (Uve, Ann), (Tom, Uve); covered $\ominus$ examples: $\emptyset$.



(a)



(b)

Table 2. Efficiency indicators.

| No Experiment | Proportion of covered $\oplus$(%) examples | Proportion of covered $\ominus$ examples (%) | Running time (sec) |
|---|---|---|---|
| 1 | 100 | 0 | 0.188 |
| 2 | 100 | 0 | 0.207 |
| 3 | 100 | 0 | 0.200 |

## Conclusion and future work

Results of presented project are:

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

2. To evaluate developed application the metrics based on three efficiency indicators are introduced.

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

2. To evaluate developed application the metrics based on three efficiency indicators are introduced.

3. Three experiments show implementation performs at sufficient efficiency.

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

2. To evaluate developed application the metrics based on three efficiency indicators are introduced.

3. Three experiments show implementation performs at sufficient efficiency.

The future steps might include:

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

2. To evaluate developed application the metrics based on three efficiency indicators are introduced.

3. Three experiments show implementation performs at sufficient efficiency.

The future steps might include:

1. Carrying out experiments with a more complex data structure, for example, when a tree node is described by a large number of predicates;

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

2. To evaluate developed application the metrics based on three efficiency indicators are introduced.
3. Three experiments show implementation performs at sufficient efficiency.

The future steps might include:

1. Carrying out experiments with a more complex data structure, for example, when a tree node is described by a large number of predicates;
2. Evaluation of the application efficiency compared to other implementations of the Foil algorithm.

## Conclusion and future work

Results of presented project are:

1. FOIL algorithm is implemented in Spark:

(i) the developed application gets the input data in extensionally form (i.e. consists of names and $\oplus$ and $\ominus$ tuples for target predicate, and names and $\oplus$ tuples for background predicates;)

(ii) the results are obtained both in intensional form (i.e. as Horn clause for the target relation that is consistent with given $\oplus$tuples and does not cover any given $\ominus$tuples) and extensional form.

2. To evaluate developed application the metrics based on three efficiency indicators are introduced.

3. Three experiments show implementation performs at sufficient efficiency.

The future steps might include:

1. Carrying out experiments with a more complex data structure, for example, when a tree node is described by a large number of predicates;

2. Evaluation of the application efficiency compared to other implementations of the Foil algorithm.

3. Testing FOIL with scoring function different from original *InformationGain* function (i.e. *Leverage* function, *LaplaceAccuracy*, etc).

## Lessons Learned

Besides the detailed study of the FOIL algorithm
we also consider as the results of this project the following:

## Lessons Learned

Besides the detailed study of the FOIL algorithm
we also consider as the results of this project the following:

1. The extension of our general skills on applications development using the Spark framework and the Scala programming language;

## Lessons Learned

Besides the detailed study of the FOIL algorithm
we also consider as the results of this project the following:

1. The extension of our general skills on applications development using the Spark framework and the Scala programming language;
2. Gaining experience in project testing and selection of evaluation datasets.

## Lessons Learned

Besides the detailed study of the FOIL algorithm
we also consider as the results of this project the following:

1. The extension of our general skills on applications development using the Spark framework and the Scala programming language;
2. Gaining experience in project testing and selection of evaluation datasets.
3. Getting deeper understanding of how and when to apply distributed means of data processing.

## References

[1] T. Horvath. Learning in logic V. In Learning from Non-Standard Data, WS 2016/17. University of Bonn, Fraunhofer IAIS, Sankt Augustin, Germany, 2016.

[2] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. The Journal of Logic Programming, 19(20):629–679, 1994.

[3] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. Machine Learning, 9(1):57–94, 1992.

[4] J.R. Quinlan. Learning logical definitions from relations. Machine Learning, (5):239–266, 1990.

[5] J.R. Quinlan and R.M. Cameron-Jones. Foil: A midterm report. In Machine Learning: ECML-93, pages 3–20. Springer-Verlag, Berlin, Heidelberg, 1993.

[6] J.R. Quinlan and R.M. Cameron-Jones. Induction of logic programs: Foil and related systems. New Generation Computing, 13(3):287–312, 1995.